1.1 ABSTRACT

The objective of the Digital combination locker is to convert a word specification to a Finite State Machine, along with that, using synchronous design methodology to derive the VHDL code for the Finite State Machine. To begin with, when the reset is asserted as 1, then state register goes within the idle state. But when, reset is set to 0, clock is in rising edge, the state register goes to the next state. In order, for the state register, which is the idle state to move to next state, i.e., C1, one should pressed enter 1 and Load, ld1= 1 to load the values, and store the values in the C1 state register. Next, to move from C1 to C2, one must pressed enter 1 and Load, ld2= 1, and store the values in the C2 state. Similarly, to move from C2to C3, one must pressed enter 1 and Load, ld3= 1, and store the values in the C3 state. Finally, to move to the unlock state, one must enter 1 and a comparison of values stored in C1, C2 and C3 are done. If the stored values are equal to 7, 5, 9, then the state register is the unlock state, else it goes to te idle state. The report contains detailed discussion and figure, on how to solve the project.

Title

A report on Combination lock system

Nabila Tabassum

40109039

A Technical Report submitted in partial fulfillment of the requirements of ENCS 281 Concordia University

April 2022

1.3 TABLE OF CONTENTS

1.4 LIST OF FIGURES

1.5 BODY OF THE REPORT

1.5.1 THE INTRODUCTION

The project aims to unlock a combination lock by using VHDL coding, with implementation of synchronous design methodology and converting the word description to Finite State Machine. The lock was provided with 3-digit combination lock, with a reset button, an enter button, with a load value ranging from ld1 to ld3.

1.5.2 THE MAIN TEXT

To begin with, the clock is in in asynchronous state, which states that when the reset is asserted as 1, then idle state goes to state register, else when the clock is at its rising edge then next state is in declared in the state register. First, when the state is in the idle state, pressed enter and load, ld1 equal 1 to move to the next state, which is the c1state.In the c1 state, values, v1 is stored in the c1 state. Similarly, pressed enter and load, ld2, ld3, to move to c2 and c3 state, additionally, store the value of v2 and v3 in the following state, c2 and c3. From c3 to go the unlock state, the state registers compare the value of v1, v2 and v3 with variables register r1, r2 and r3. In the variable registers, a constant value for r1, r2 and r3 of 4 bit is declared as “0111”, “0101” and “1001” is declared. A comparison of values and variable registers is done, If the

values are equal then the next state goes to the unlock state, which is represented by declaring 1 in the unlock state. If the value is not equal, then the state register goes to idle state.

1.5.3 THE CONCLUSION

A successful implementation of digital combinational locker is done. The code for the digital combinational locker was compiled successfully in VHDL and a .do file was implemented to compile the result for the following. The modelsim output, to gain the desired result is attached below.

```
VSIM 2> quit
[murphy] [/home/n/n_tabas/COEN313/Code] > vsim -c -do comb_lock.do comb_lock
Reading /nfs/sw_cmc/x86_64.EL7/tools/mentor.2011/modelsim_6.6g/modeltech/tcl/vsim/pref.tcl

# 6.6g

# vsim -do comb_lock.do -c comb_lock
# //  ModelSim SE-64 6.6g May 23 2012 Linux 3.10.0-1160.45.1.el7.x86_64
# //
# //  Copyright 1991-2012 Mentor Graphics Corporation
# //             All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Loading work.comb_lock(multi_seg_arch)#1
# do comb_lock.do
# idle 1 0 0 1 0
# c1 0 1 0 1 0
# c2 0 0 1 1 0
# c3 0 0 0 1 0
# end_state 0 0 0 1 1
# idle 0 0 0 1 0
#
#
VSIM 2>
```

Fig 1: Screenshot of the modelsim, containing the results.

The .vhd Code and .do file is attached in the appendix.

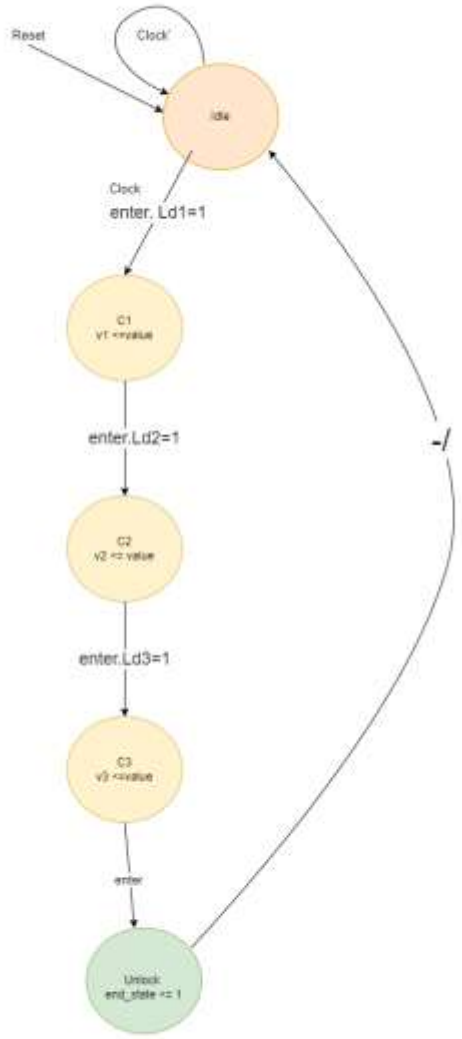

Fig 3: FSM chart of the Controller

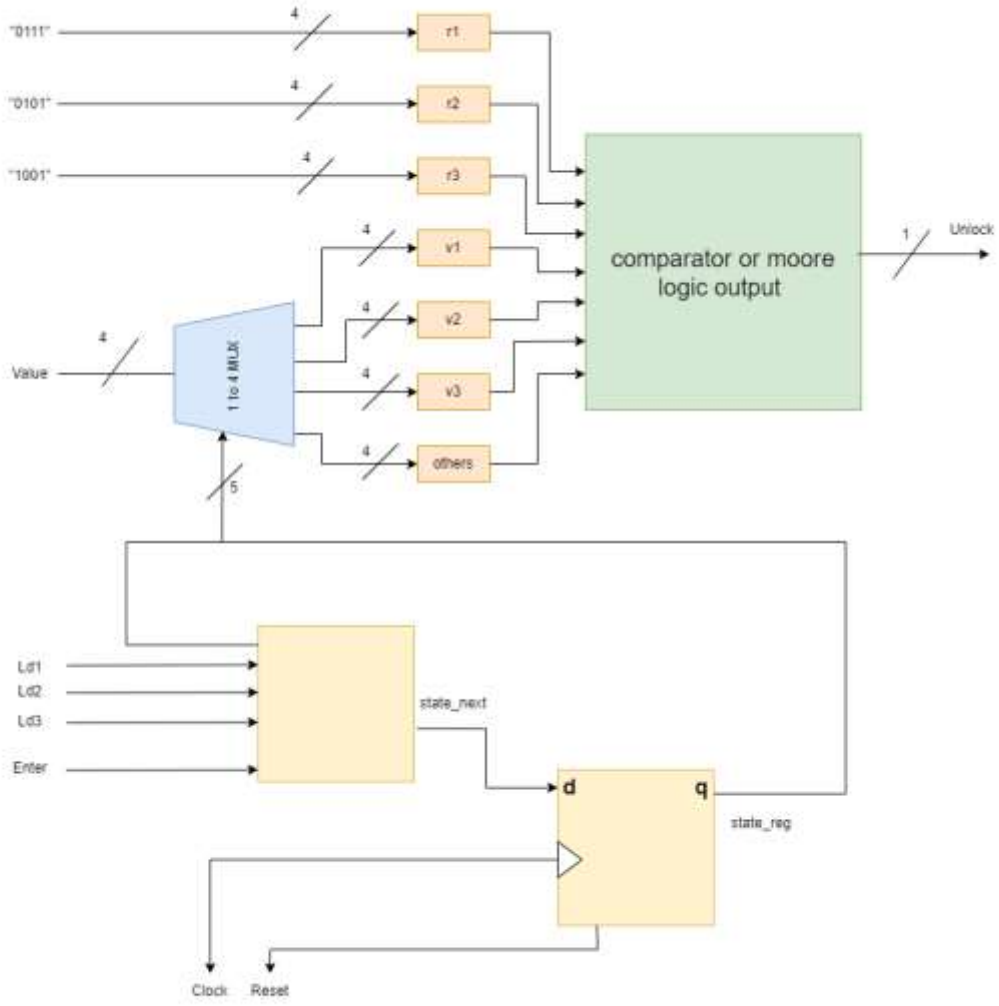

Fig3: Conceptual diagram of the whole lock process.

1.6 LIST OF REFERENCE

- RTL hardware design using VHDL [**Pong P. Chu, 1959**]

1.7 APPENDIX

--comb_lock.vhd;

```
Library IEEE;
USE IEEE.std_logic_1164.all;
ENTITY comb_lock IS
 PORT(
  ld1 : IN STD_LOGIC;
 ld2 : IN STD_LOGIC;
 ld3 : IN STD_LOGIC;
 value : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
 enter : IN STD_LOGIC;
 reset : IN STD_LOGIC;
 clock : IN STD_LOGIC;
 unlock : Out STD_LOGIC);
end comb_lock;

architecture multi_seg_arch of comb_lock is
type mc_state_type is
( idle,c1,c2,c3, end_state);
signal state_reg, state_next: mc_state_type;
signal v1, v2, v3 : std_logic_vector(3 downto 0);
begin

--state register
process(clock, reset)
begin
if (reset = '1')then
  state_reg <= idle;
  elsif ( clock'event and clock='1') then
  state_reg <= state_next;
  end if;
  end process;

  --next-state logic
  process ( state_reg, enter, ld1, ld2, ld3   )
  begin
    case state_reg is
    when idle =>
       if enter = '1' then
```

```
          if ld1= '1' then
               state_next <= c1;
           else
                state_next <= idle;
                end if;
               end if;
             when c1 =>
                  if enter = '1' then
                      if ld2 ='1' then
                          state_next <= c2;
                        end if;
                    end if;
              when c2 =>
                    if enter = '1' then
                       if ld3 ='1' then
                         state_next <= c3;
                       end if;
                      end if;
             when c3 =>
                  if enter = '1' then
                      state_next<= end_state;
                        else
                       state_next <= idle;
                      end if;

           when others =>
                                          state_next<=idle;

                end case;
        end process;

--moore state logic
     process ( state_reg )
     begin

      case state_reg is
         when idle =>
         when c1 =>
         v1 <= value;
         when c2 =>
         v2 <= value;
         when c3 =>
         v3 <= value;
         when others =>
```

```
        end case;

        end process;

   --comaparator OR MOORE OUTPUT LOGIC


        process(state_reg)


   variable r1 : std_logic_vector(3 downto 0):= "0111";
              variable r2 : std_logic_vector(3 downto 0):= "0101";
   variable r3 : std_logic_vector(3 downto 0):= "1001";
         begin
           unlock <= '0';
                                              if (state_reg = end_state) then
                              if (r1=v1 and r2=v2 and r3=v3) then
            unlock<= '1';

            end if;
          end if;


               end process;

end multi_seg_arch;
```

--.do file

```
force clock 0

run 2


force clock 1
force reset 0
force ld1 1
force ld2 0
force ld3 0
force enter 1


run 2

examine state_reg ld1 ld2 ld3 enter unlock

force clock 0
run 2



force clock 1
force reset 0
force ld1 0
force ld2 1
force ld3 0
force enter 1
force value 0111


run 2

examine state_reg ld1 ld2 ld3 enter unlock

force clock 0
run 2
```

```
force clock 1
force reset 0
force ld1 0
force ld2 0
force ld3 1
force enter 1
force value 0101


run 2

examine state_reg ld1 ld2 ld3 enter unlock

force clock 0
run 2


force clock 1
force reset 0
force ld1 0
force ld2 0
force ld3 0
force enter 1
force value 1001

run 2

examine state_reg ld1 ld2 ld3 enter unlock

force clock 0
run 2

force clock 1
force reset 0
force ld1 0
force ld2 0
force ld3 0
force enter 1

run 2

examine state_reg ld1 ld2 ld3 enter unlock

force clock 0
```

run 2

force clock 1
force reset 0
force ld1 0
force ld2 0
force ld3 0
force enter 1

run 2

examine state_reg ld1 ld2 ld3 enter unlock